

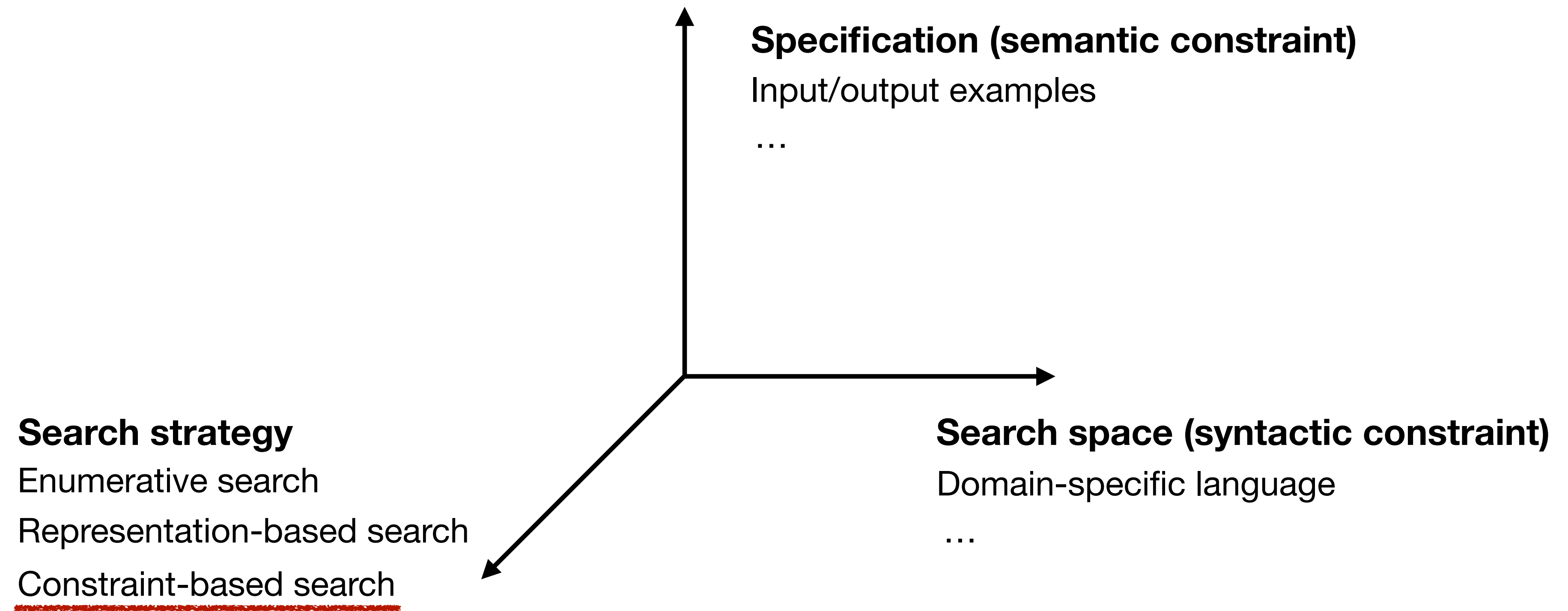
Program Reasoning

14. Constraint-based Search

Kihong Heo



Dimensions in Program Synthesis



Application: Programming with APIs

- Synthesizing a program using a given set of APIs (e.g., java.awt.geom libraries)

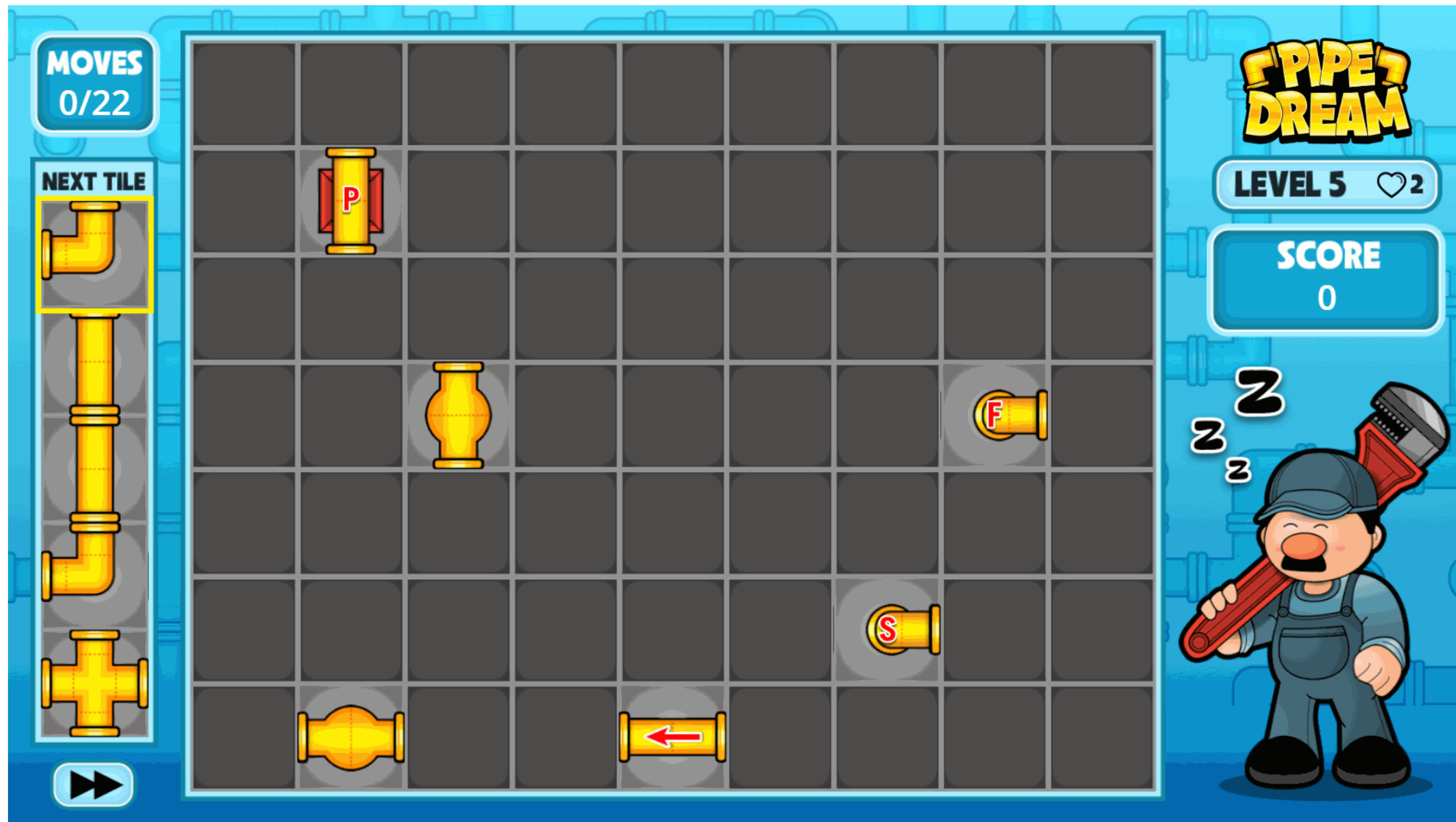
```
Area rotate(Area obj, Point2D pt, double angle) {  
    AffineTransform at = new AffineTransform();  
    double x = pt.getX();  
    double y = pt.getY();  
    at.setToRotation(angle, x, y);  
    Area obj2 = obj.createTransformedArea(at);  
    return obj2;  
}
```

Application: Bit-twiddling

- Synthesizing a program using a given set of bitwise operators

```
// Round up to the next
// highest power of 2
fun f(x):
    o1 = bvsb(x, 1)
    o2 = bvshr(o1, 1)
    o3 = bvor(o1, o2)
    o4 = bvshr(o3, 2)
    o5 = bvor(o3, o4)
    o6 = bvshr(o5, 4)
    o7 = bvor(o5, o6)
    o8 = bvshr(o7, 8)
    o9 = bvor(o7, o8)
    o10 = bvshr(o9, 16)
    o11 = bvor(o9, o10)
    o12 = bvadd(o11, 1)
    return o12
```

Pipe Dream



Constraint-based Search

- Idea: encode the synthesis problem as a constraint-solving problem (SAT/SMT)
 - Constraints: syntactic and semantics constraints of the program
 - Program search: proof search by the solver
 - Solution program: directly derived from the solution of the SAT/SMT problem
- Target programs
 - Loop-free programs
 - Composition of functions that can be encoded as SMT formula

Target Program

- Given a finite multiset of components $\{\text{component}_1, \dots, \text{component}_n\}$,

```
fun synthesize_program(inputs, ...):  
  tempi = componenti(parama, ...)  
  tempj = componentj(paramb, ...)  
  ...  
  tempk = componentk(paramk, ...)  
  return tempk
```

- E.g., synthesize $f: A \times A \rightarrow C$ using components $\{g, g, h\}$ where $g: A \rightarrow B$ and $h: B \times B \rightarrow C$

```
fun f(x, y):  
  tmp0 = g(x)  
  tmp1 = h(tmp0, tmp0)  
  return tmp1
```



```
fun f(x, y):  
  tmp0 = g(x)  
  tmp1 = g(y)  
  tmp2 = h(tmp0, tmp1)  
  return tmp2
```



```
fun f(x, y):  
  tmp0 = g(x)  
  tmp1 = h(tmp0, tmp0)  
  tmp2 = h(tmp0, tmp0)  
  return tmp2
```



Example

- Bitvector manipulation program $f : \text{BitVec} \rightarrow \text{BitVec}$
- Specification: $f(01100) = 01000 \wedge f(10001) = 10000$
 - Intention: replace the rightmost 1 with 0
- Components:
 - $f_1(a) = a - 1$
 - $f_2(a, b) = a \& b$
- Solution: $f(x) = x \& (x - 1)$

Program as DAG

Component library:

$f_1(a) = a - 1$

$f_2(a, b) = a \ \& \ b$

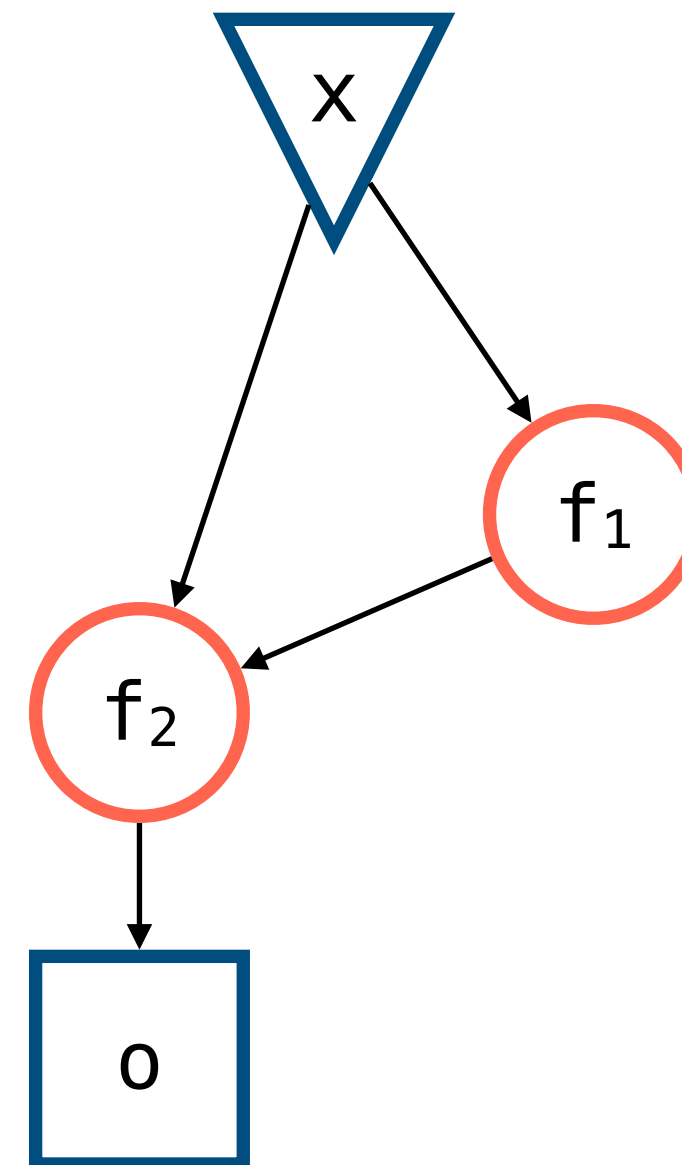
Solution program:

fun $f(x)$:

$o_1 = f_1(x)$

$o_2 = f_2(x, o_1)$

return o_2



**How to represent the correct edges
between given nodes?**



Program Location as Number

Component library:

$$f_1(a) = a - 1$$

$$f_2(a, b) = a \ \& \ b$$

Solution program:

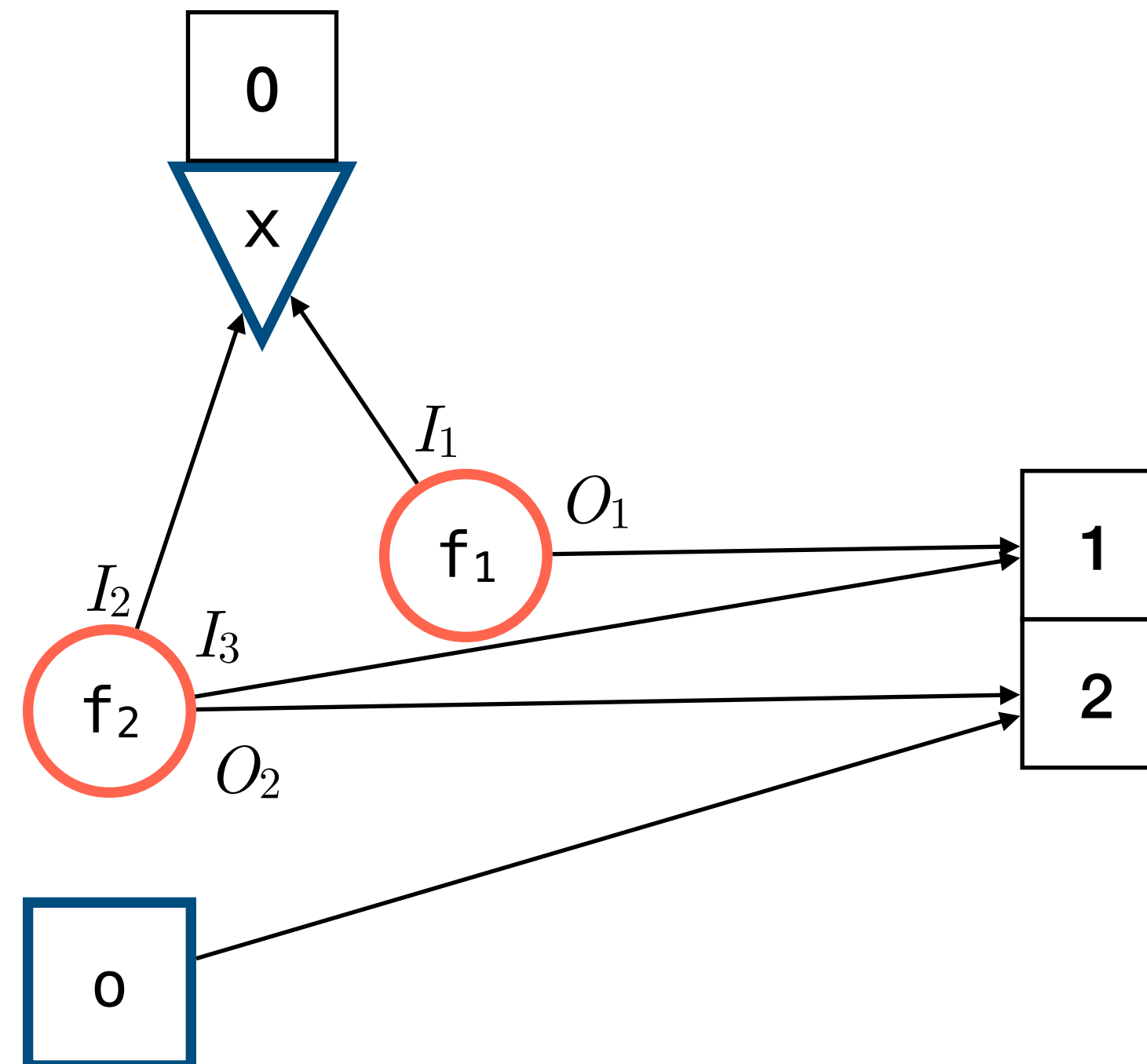
```
0: fun f(x):  
1:   o1 = f1(x)  
2:   o2 = f2(x, o1)  
   return o2
```

The solution program corresponds to the following assignment:

$$l_{I_1} = 0 \quad l_{O_1} = 1$$

$$l_{I_2} = 0 \quad l_{O_2} = 2$$

$$l_{I_3} = 1$$

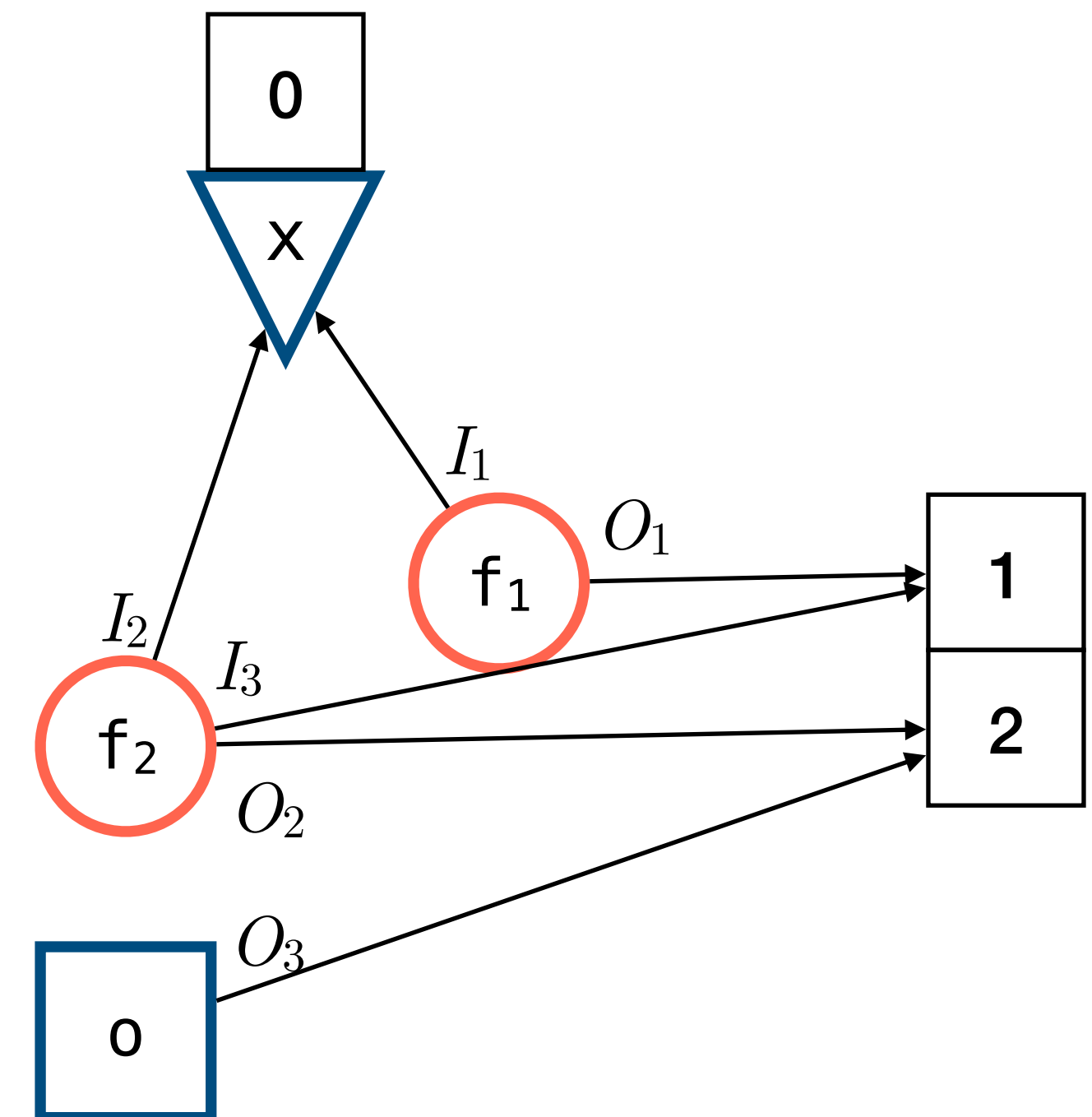


How to make an SMT solver find this assignment?



SMT Encoding (1): Variables

- Input variables: represent the inputs of each component
 - $P = \{I_1, I_2, I_3\}$
- Output variables: represent the outputs of each component
 - $R = \{O_1, O_2\}$
- Location variables: represent the location of each input and output
 - $L = \{l_x \mid x \in P \cup R\}$



SMT Encoding (2): Well-formedness

- A program is well-formed if and only if

$$\psi_{\text{wfp}} = \bigwedge_{x \in P} (0 \leq l_x \leq 2) \wedge \bigwedge_{x \in R} (1 \leq l_x \leq 2) \wedge \psi_{\text{cons}} \wedge \psi_{\text{acyc}}$$

Locations of the inputs
of each component

Locations of the outputs
of each component

Consistency constraint

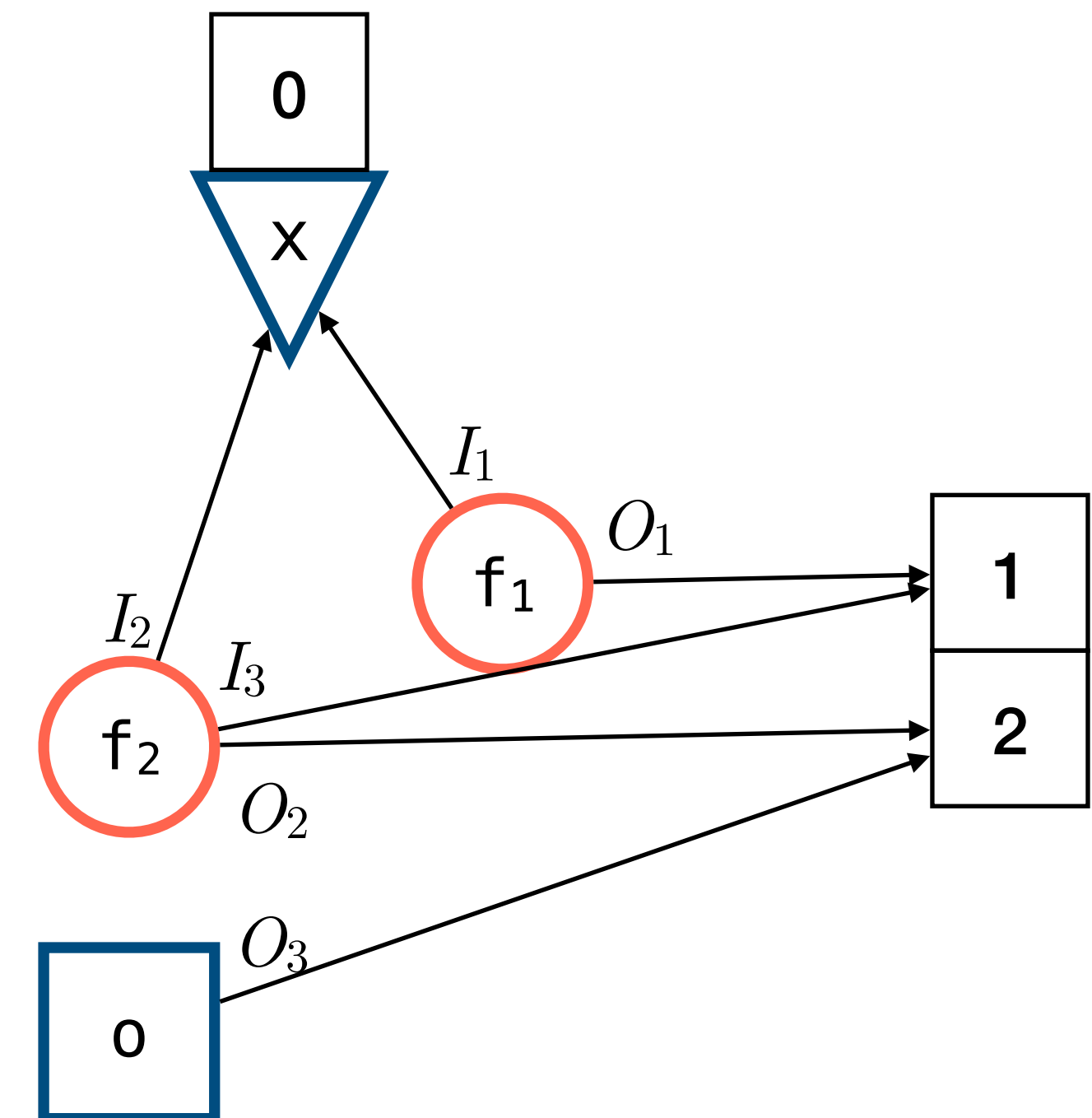
$$\psi_{\text{cons}} = l_{O_1} \neq l_{O_2}$$

At most one component
per each line

Acyclicity constraint

$$\psi_{\text{acyc}} = l_{I_1} < l_{O_1} \wedge l_{I_2} < l_{O_2} \wedge l_{I_2} < l_{O_2}$$

No uninitialized variable



SMT Encoding (3): Semantics of Components

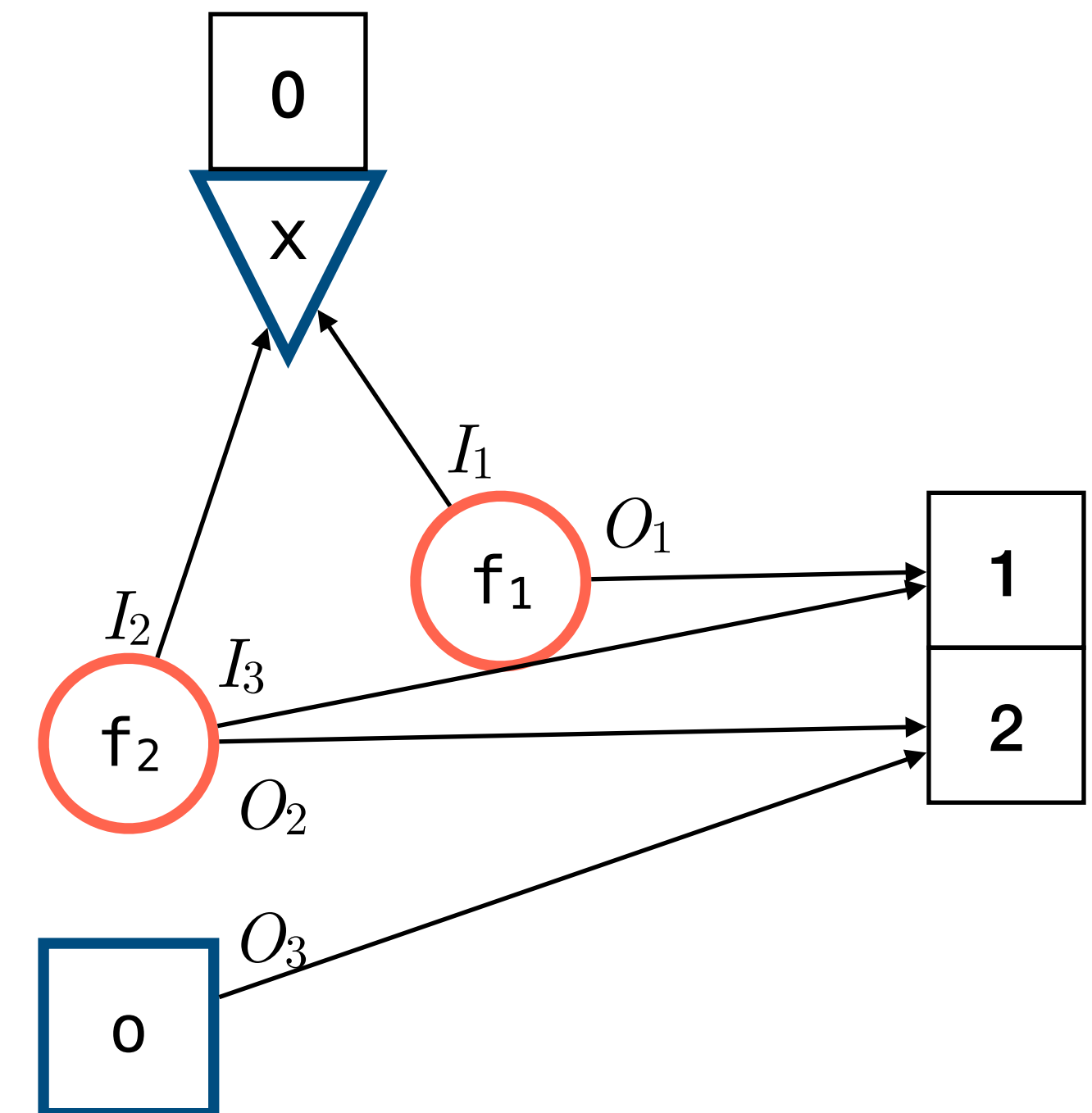
- A program consists of a given set of components

Component library:

$$f_1(a) = a - 1$$

$$f_2(a, b) = a \ \& \ b$$

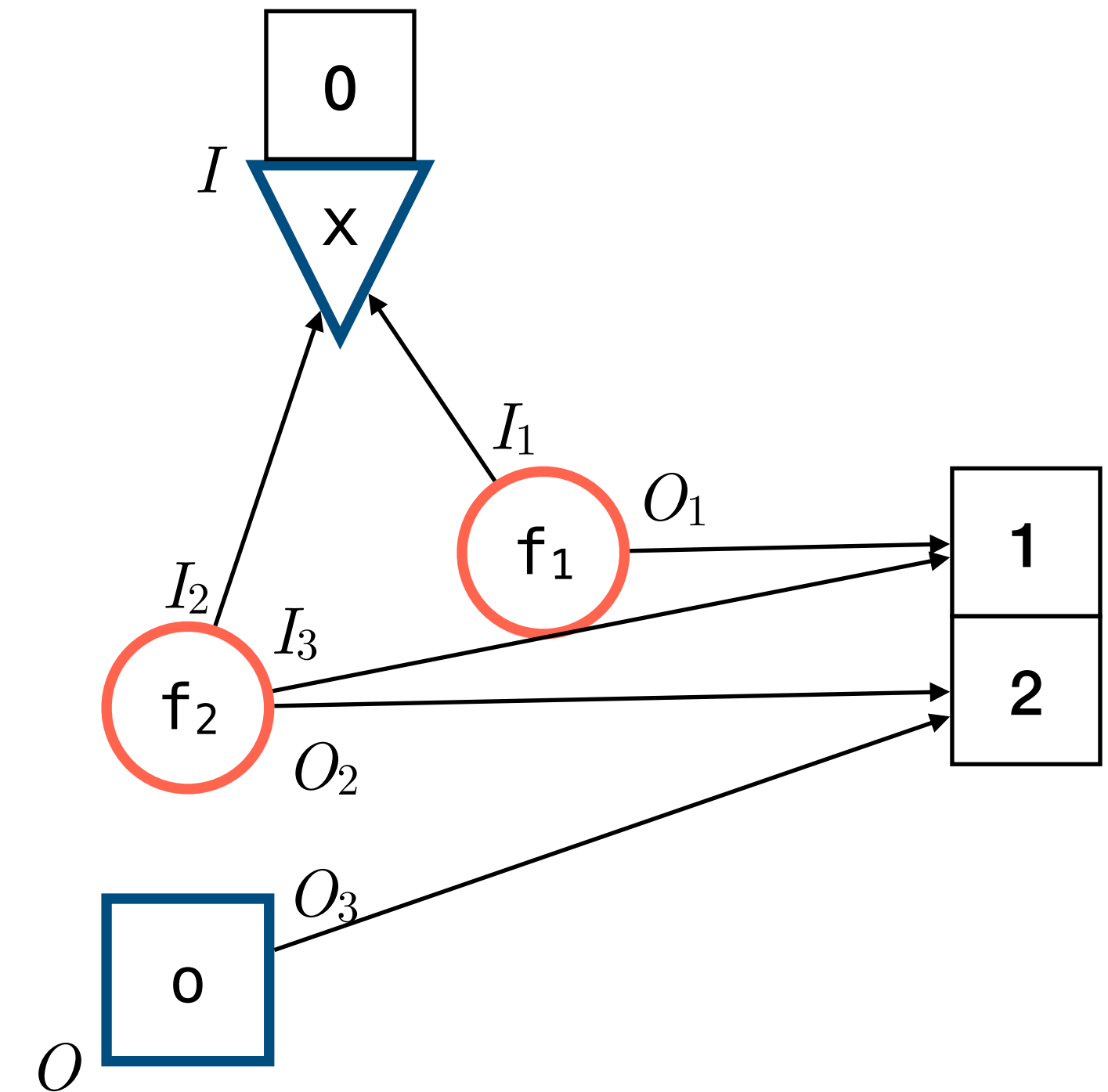
$$\psi_{\text{lib}} = (O_1 = I_1 - 1) \wedge (O_2 = I_2 \ \& \ I_3)$$



SMT Encoding (4): Dataflow

- A program consists of data-flows

$$\psi_{\text{conn}}(I, O) = \bigwedge_{x, y \in P \cup R \cup \{I, O\}} (l_x = l_y \implies x = y)$$



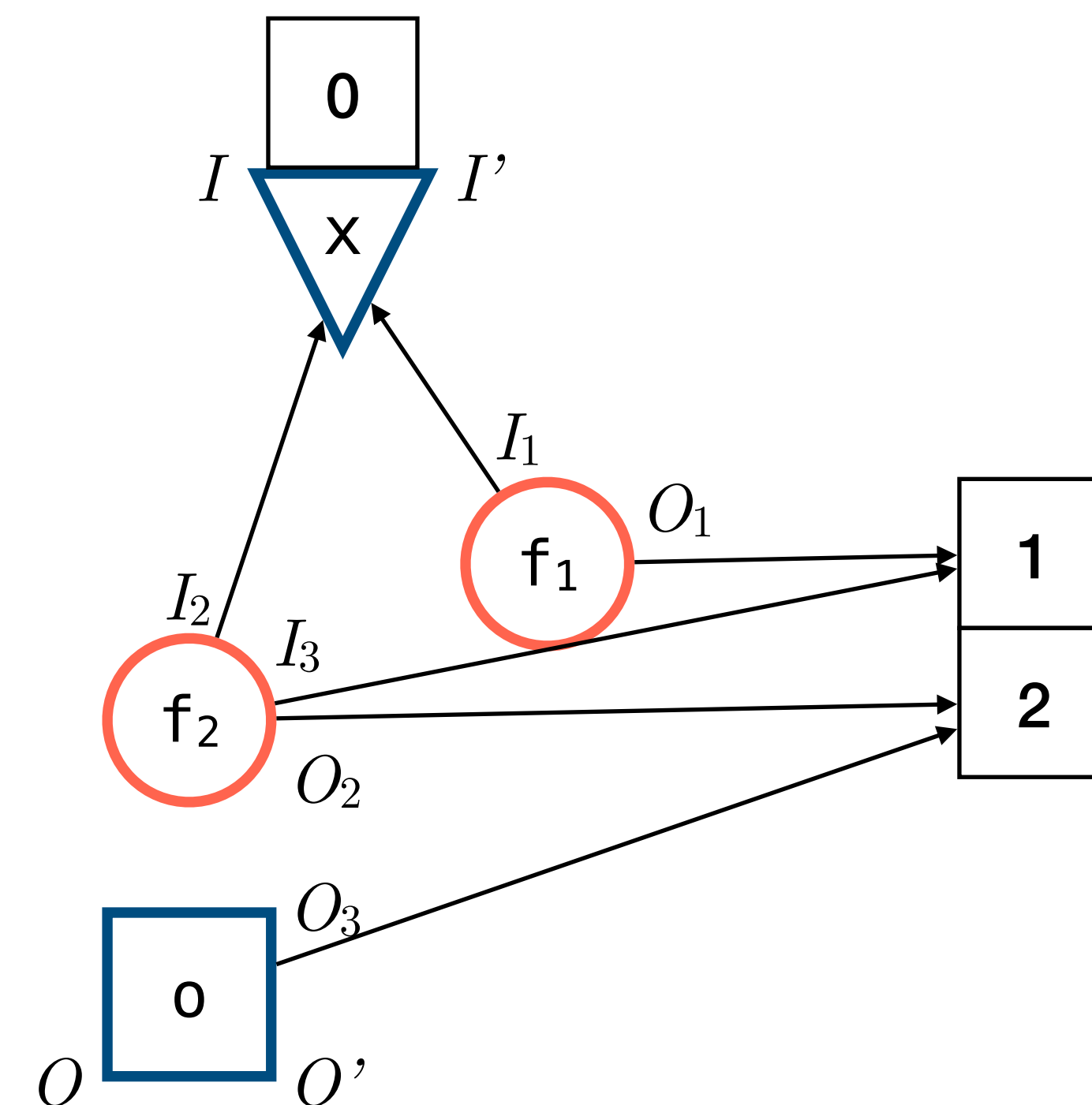
Putting All Together

- Specification: $f(01100) = 01000 \wedge f(10001) = 10000$
 - Intention: replace the rightmost 1 with 0
- Components: $f_1(a) = a - 1$ and $f_2(a, b) = a \& b$
- Solution: assignments of variables l_x satisfying formula F

$$F = \psi_{\text{wfp}} \wedge \psi_{\text{lib}} \wedge F_1 \wedge F_2 \quad \textbf{where}$$

$$F_1 : I = 01100 \wedge O = 01000 \wedge \psi_{\text{conn}}(I, O)$$

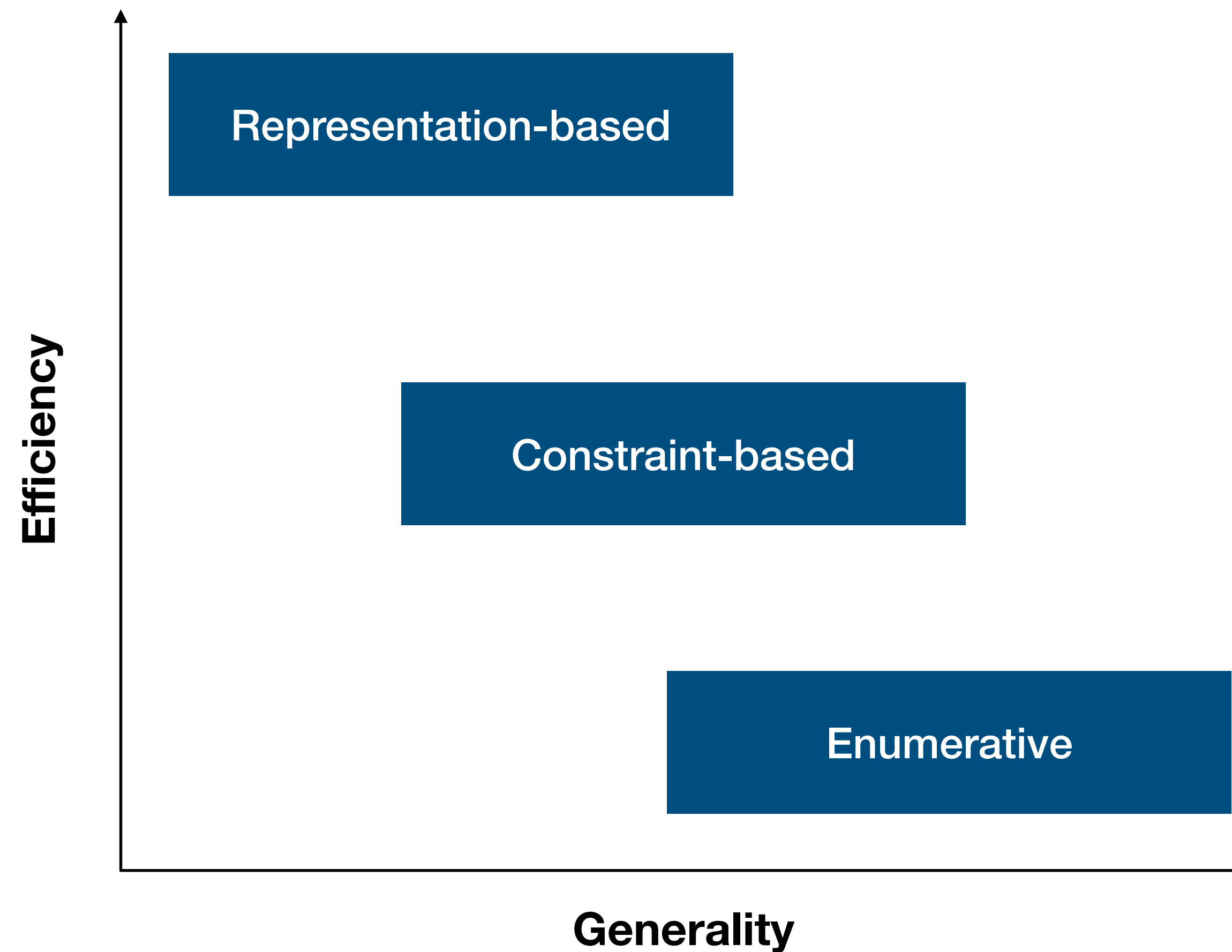
$$F_2 : I' = 010001 \wedge O' = 10000 \wedge \psi_{\text{conn}}(I', O')$$



Properties

- Decisive performance factor: # components
- Depending on the performance of SMT solvers (gradually improving)
- Multiplicity constraints: “must use some operator $\leq n$ times”
 - Not easy to specify such syntactic constraints using CFG
 - Applicable to apply to synthesize resource-sensitive programs (e.g., use multiplication operator up to 3 times in homomorphic encryption scheme)

Comparison of Search Strategies



Summary

- Constraint-based search: encode synthesis problems as constraint-solving problems
- SMT encoding: syntactic constraints \wedge semantic constraints
- Application: API programming, bit-twiddling, resource-sensitive programming